Planned learning curve and some initial programming ideas to get us started:

I. Setting Up the Development Environment and LimeSDR

* Install Lime Suite: This is the software library for interfacing with LimeSDR devices. You'll need this to control the SDR from your C++ code.
* C++ Setup: Ensure you have a working C++ development environment (e.g., Visual Studio, g++, CMake).
* Familiarize Yourself with the LimeSDR Mini 2.0: Understand its basic specifications (frequency range, bandwidth, etc.).

II. Basic SDR Control with C++

* Connecting to the LimeSDR: Learn how to establish a connection to the LimeSDR Mini 2.0 using the Lime Suite API.
* Configuring the SDR: Write C++ code to set the center frequency, sample rate, and bandwidth of the LimeSDR.
* Transmit and Receive: Implement basic transmit and receive functionality to send and receive simple waveforms (e.g., sine waves).

III. FMCW Chirp Signal Generation and Transmission

* Chirp Signal Generation:
  + Implement a function in C++ to generate the FMCW chirp signal. This involves creating a time-varying sinusoidal waveform where the frequency changes linearly with time.
  + Key parameters for a chirp signal are:
    - Start frequency
    - Stop frequency
    - Chirp duration
    - Sample rate
* Transmission: Modify your transmit code to send the generated chirp signal via the LimeSDR's TX port.

IV. Signal Reception and Processing

* Receiving Signals: Configure the LimeSDR to receive the reflected signals on the RX port.
* Signal Conditioning:
  + Implement any necessary signal conditioning steps (e.g., filtering, amplification).
* RSSI Calculation:
  + Calculate the Received Signal Strength Indicator (RSSI) from the received signal samples. This usually involves computing the average power of the signal over a short period.
* RTT Calculation:
  + Calculate the Round-Trip Time (RTT) by analyzing the time delay between the transmitted and received chirp signals.
  + This often involves techniques like correlation to find the time shift between the signals.
* Distance Calculation:
  + Convert the calculated RTT to distance using the speed of light.

V. Distance-Based Filtering

* Implement Filtering Logic:
  + Create a mechanism to filter the received signals or the calculated RTT/distance values based on your specified distance range.
  + This might involve setting thresholds and discarding data outside the range of interest.

VI. Material Analysis and Visualization

* Feature Extraction:
  + Extract relevant features from the filtered received signals that might provide information about the material in the wave path.
  + Examples: changes in signal amplitude, frequency response, or phase shifts.
* Visualization:
  + Use a C++ plotting library (e.g., matplotlib-cpp) to create visualizations of the extracted features.
  + This could involve:
    - Graphs of signal strength vs. time
    - Frequency spectra
    - Plots of distance vs. signal characteristics

Important C++ and SDR Concepts

* Digital Signal Processing (DSP): You'll be using DSP techniques extensively for chirp generation, RSSI calculation, RTT estimation, and feature extraction.
* Data Buffering and Streaming: Efficiently handling the flow of data between the LimeSDR and your C++ application is crucial.
* Multi-threading: Consider using multi-threading to handle simultaneous transmit and receive operations or to perform processing in parallel.
* Error Handling: Implement robust error handling to deal with potential issues during SDR operation.